

# APPENDIX

---

## Special Features of Atari Logo

---

Atari Logo takes advantage of the unique hardware design of Atari home computers. It has features such as dynamic turtles, a sound generator, and detectors for turtles bumping into one another or bumping into lines drawn on the screen. Many projects in this book use these features.

---

### Turtle Graphics

Like most Logos, Atari Logo has a graphics turtle that has a drawing pen and can leave a path as it moves and turns about the screen. The special hardware of Atari computers allows Atari Logo to have four turtles. These turtles are dynamic. They can travel about the screen at different speeds and, if their pens are down, they draw and leave a trace of their paths. If they are going fast, their paths tend to be dotted rather than solid lines.

Color is another way in which Atari Logo differs from other Logos. Atari Logo has 128 different colors. The colors range through sixteen hues, each of which consists of eight different intensities (luminances). These colors are referred to by number from 0 through 127. Any of these colors can be used for the background, the lines the turtles draw, or the turtles themselves.

#### *Turtle Color and Speed*

Each of the four turtles has been given a different color for its shape when it starts. You can change the color of a turtle's shape to any of the 128 colors. In the following example the turtles head in different directions, but travel at the same speed and periodically change color.

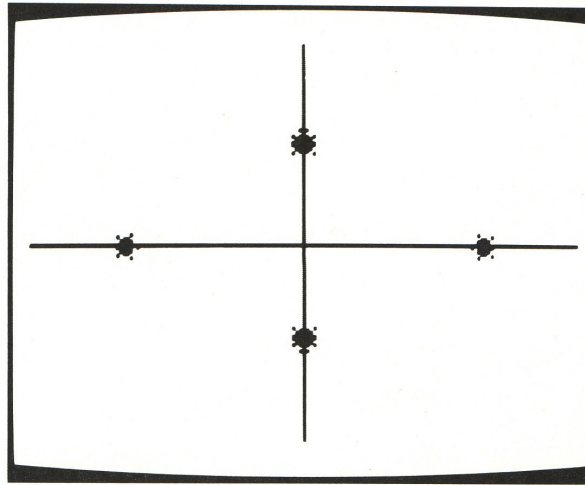
```

TO FLIP
  TELL [0 1 2 3]
  CS ST PD FS
  EACH [RT 90 * WHO]
  SETSP 30

  REPEAT 10 [WAFFLE]
  RESET
  SS WAIT 180
  CS HT TELL 0 ST
  END

  TO WAFFLE
    EACH [SETC RANDOM 128]
    WAIT 80
  END

```



RESET restores turtle 0 to 7 (white), turtle 1 to 20 (orange), turtle 2 to 44 (purple), and turtle 3 to 60 (blue).

```

TO RESET
  ASK 0 [SETC 7]
  ASK 1 [SETC 20]
  ASK 2 [SETC 44]
  ASK 3 [SETC 60]
  END

```

The Logo operation that outputs the current color of a turtle is COLOR. To find the current speed of a turtle, use SPEED.

### *Pens and Pen Color*

At any one time the lines drawn by turtles on the screen are limited to three different colors. This limitation is imposed because there are only three pens: pen 0, pen 1, and pen 2. Each pen can write in any of the 128 different colors. When you change the color in a pen, lines to be drawn by that pen and those already on the screen will appear in that color. This feature of being able to change the color in a pen and thus change the color of lines on the screen after they have been drawn leads to some startling graphics effects.

When you first start up Logo, the turtles are all using pen 0, which contains pen color 15 (yellow). If you want a turtle to use a different pen, then you type SETPN 1 or SETPN 2. To change the pen color you use the command SETPC. For example, SETPC 0 7 makes the color in pen 0 become white. SETPC 0 15 changes pen 0 back to yellow. PN is the Logo operation that outputs the current pen; PC outputs the color of whatever pen you give as input.

If you refer to a pen with a number other than 0, 1, or 2, Logo will print out an error message. Logo will also complain if you refer to a color with a number outside the range 0 through 127.

*Using One Pen*

In the following example, the turtle makes a star.

```
TO STAR :SIZE
REPEAT 6 [LEG :SIZE RT 60]
END
```

```
TO LEG :SIZE
PU FD :SIZE / 3 PD
FD :SIZE BK :SIZE
PU BK :SIZE/3 PD
END
```

Now try:

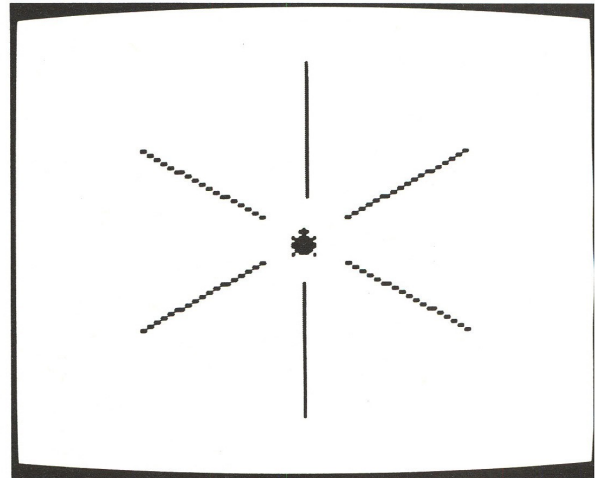
```
STAR 30
```

and then type

```
REPEAT 20 [SETPC 0 RANDOM 128 WAIT 20]
```

You can use more than one turtle to make stars. For example:

```
TO THREESTARS
TELL [0 1 2] CS ST
EACH [RT WHO * 120]
PU FD 60
STAR 30
END
```



Now:

```
THREESTARS
REPEAT 20 [SETPC 0 RANDOM 128 WAIT 20]
```

*Using Three Pens*

In the previous example all three turtles used the same pen, so the three stars change to the same color. But each of the three turtles can use a different pen. To do this, change THREESTARS or make a new procedure like STAR3.



```

TO STAR3
TELL [0 1 2] CS
EACH [RT WHO * 120 SETPN WHO]
PU FD 60
STAR 30
END

```

Try:

```

STAR3
REPEAT 10 [EACH [SETPC WHO RANDOM 128 WAIT 20]]

```

Try WHIRL after running STAR3.

```

TO WHIRL
SETPC 0 56
SETPC 1 56
SETPC 2 56
WHIRL1 0 7 56
END

```

```

TO WHIRL1 :PN :NEWC :OLDC
IF :PN = 3 [SETPN 0 SETPC 0 :OLDC STOP]
SETPC :PN :NEWC
WAIT 15
SETPC :PN :OLDC
WHIRL1 :PN + 1 :NEWC :OLDC
END

```

Type:

```

REPEAT 3 [WHIRL]

```

When you finish, you might want to reset the colors in the pens and have all the turtles use pen 0.

```

TO RESETPC
SETPC 0 15
SETPC 1 47
SETPC 2 121
ASK [0 1 2 3] [SETPN 0]
END

```

For examples of different ways of using pens, look at projects such as *Cartoon* and *Animating Line Drawings*.

### ***Background Color***

Here is a program that changes the background color from 0 through 127. Before RAINBOW stops, it sets the background back to the starting color, 74.



## SPECIAL FEATURES OF ATARI LOGO

```

TO RAINBOW :DELAY
FS
SETBG 0 WAIT :DELAY
RAINBOW1 :DELAY
SETBG 74 SS
END

```

In running through the background colors, an artifact of standard television signals appears. The TV picture jumps whenever you go from very bright to very dark. You can see this effect by typing:

```

SETBG 7
SETBG 8

```

RAINBOW avoids this by skipping over fifteen color changes, where the shift is from very bright to very dark. These occur at transitions between hue boundaries going from 7 to 8, 15 to 16, 23 to 24, 31 to 32, 39 to 40, 47 to 48, and so on. RAINBOW1 skips over the darkest boundary colors 8, 16, 24, 32, and other multiples of 8 through 120.

```

TO RAINBOW1 :DELAY
IF BG > 126 [STOP]
IF 0 = REMAINDER BG + 1 8 [SETBG BG + 2] [SETBG BG + 1]
WAIT :DELAY
RAINBOW1 :DELAY
END

```

Try:

```
RAINBOW 30
```

Changing the background while there are drawings on the screen or turtles in motion can lead to some compelling graphics displays.

## Turtles and Their Shapes

### *Designing a Shape*

All the turtles start as the same shape (that of a land turtle), but you can design other shapes yourself. Logo sets aside memory just for these shapes. We refer to the shape memory by one of sixteen slot numbers. Slots 1 through 15 are available for your own shapes. Slot 0 always contains Logo's regular turtle shape. The Logo operation `SHAPE` outputs the number of the shape the turtle currently carries. `SETSH` is the command by which the turtle's shape can be changed to any of the ones in slots 0 through 15.

You can make shapes in the shape editor. For example, when you type:

---

By Cynthia Solomon.

## EDSH 1

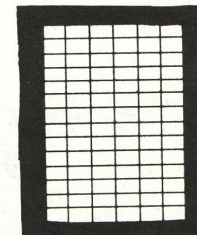
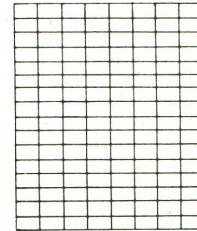
the shape editor displays a grid of eight columns and sixteen rows.

At the start, the outline of the upper left rectangle is black while the rest of the grid is white. The black rectangle is the cursor that you move around the grid using the arrow keys (with the CTRL key).

You make a design by filling in cells of the grid. To fill in a cell (a rectangle), move the cursor there and then press the space bar. To clear a cell you also move the cursor to the cell and press the space bar. Notice that pressing the space bar does not move the cursor.

Let's make a shape; fill in the edges of the grid to make a hollow rectangle.

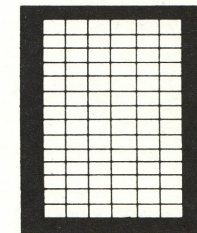
You get out of the shape editor the same way you do the text editor, by pressing the ESC key.\*

*Using Shapes*

Now change the shape of, for example, turtle 0. You might want to make sure that you are talking to turtle 0 and that it is visible.

```
TELL 0
ST
SETSH 1
```

Sometimes people say that a turtle is "carrying" a particular shape. So, for example, turtle 0 is now carrying shape 1.

*Animation*

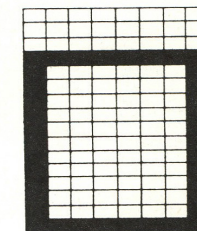
You can use the shape you just made in an animation sequence where the shape shrinks and grows. For this you need to make another shape. Make a smaller rectangle as shape 2.

Type:

## EDSH 2

By alternating the two shapes with a delay after the changes, you have a box that shrinks.

```
TO SHRINK
SETSH 1
WAIT 10
SETSH 2
WAIT 10
END
```



When SHRINK is repeated, the box shrinks and grows continually. For example, type:

\*If you press the BREAK key, you will lose whatever changes you made during this editing session.

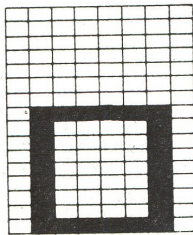


```
REPEAT 20 [SHRINK]
```

### *Another Animation*

Make a smaller box and extend SHRINK to include it. You might type EDSH 3 to do this. After making the new shape, change SHRINK to include as its last two instructions

```
SETSH 3
WAIT 10
```



Now when you try REPEAT 20 [SHRINK], the smooth effect of shrinking and growing has been upset. Another step is needed. Rather than changing SHRINK, make a new procedure.

```
TO STRETCH
SHRINK
SETSH 2
WAIT 10
END
```

```
REPEAT 20 [STRETCH]
```

GROW has all four turtles doing STRETCH.

```
TO GROW
TELL [0 1 2 3] CS PU ST
SETBG 0 FS
EACH [RT WHO * 90]
SETSP 20
REPEAT 20 [STRETCH]
CS SETSH 0
SETBG 74 SS
END
```

### *The Regular Turtle Shape, Shape 0*

At any time you can change the turtle back to shape 0 by typing SETSH 0. A turtle always starts carrying shape 0. You cannot change what is in this slot. Nor can you view this in the shape editor. Try:

```
SETSH 0
CS
TELL 0
REPEAT 24 [RT 15 WAIT 10]
```

As the turtle turns, its *body* turns and changes. Shape 0 changes at 15-degree intervals. Thus, if a turtle's heading is 0 and you turn it right 10 degrees, its new heading will be 10 degrees, but the shape it carries will not have turned or changed. Turn the turtle 5 more degrees and the shape will show a change of rotation.



*Rotating Shapes*

There is an important difference between Logo's regular turtle shape and the shapes you make. The rectangle shape you have just made does not rotate. If you want a shape you have made rotate, you have to design the shape in its different orientations and specifically order the turtle to assume that new shape after each rotation. (See the Pacgame and Blaster projects.)

*Keeping and Reusing Shapes*

When you reboot Logo, the shapes you made are not in the shape memory; slots 1 to 15 are cleared out. (Notice that ERALL does not clear out these slots.) If you want to use GROW or STRETCH again, you have to put the three shapes into the same slots they were in before. Thus you have to figure out a way to save them and also a way to load them.

There are two steps to saving shapes. The first is getting the information from the shape memory into your workspace. Using the Logo operation GETSH, you can get a description of each shape. For example, GETSH 1 will output a list of numbers that describes the large rectangle. To put the three rectangle shapes in your workspace, you make three variables.

```
MAKE "SHAPE1 GETSH 1
MAKE "SHAPE2 GETSH 2
MAKE "SHAPE3 GETSH 3
```

:SHAPE1, :SHAPE2, and :SHAPE3 are now lists of sixteen numbers.

The second step can be performed at this time. Now you can save these shapes on diskette or cassette using the command SAVE. SAVE writes your entire workspace onto diskette, cassette, or paper. If you want to save only these variables, you will have to clean out your workspace.

Although you can put these variables into your workspace either by typing them in again or by loading them from diskette or cassette, you must still put them back into their appropriate slots. To do this use the Logo command PUTSH. PUTSH takes two inputs: the first is the slot and the second is a description of a shape.

For example, SETUP is one way to ensure that the shapes are set up.

```
TO SETUP
  PUTSH 1 :SHAPE1
  PUTSH 2 :SHAPE2
  PUTSH 3 :SHAPE3
END
```

Then put SETUP in GROW so that the shapes are in the slots before STRETCH is run.

In this book there are examples of other methods to save and load shapes. Here are two procedures that will do the job.

## SPECIAL FEATURES OF ATARI LOGO

```

TO KEEPSHAPES :SLOTNUM
IF :SLOTNUM = 0 [STOP]
MAKE WORD "SHAPE :SLOTNUM GETSH :SLOTNUM
KEEPSHAPES :SLOTNUM - 1
END

```

KEEPSHAPES 3 will put shapes 1 through 3 in the variables SHAPE1, SHAPE2, and so forth.

```

TO PUTSHAPES :SLOTNUM
IF :SLOTNUM = 0 [STOP]
PUTSH :SLOTNUM THING WORD "SHAPE :SLOTNUM
PUTSHAPES :SLOTNUM - 1
END

```

SETUP can be changed to use PUTSHAPES.

```

TO SETUP
PUTSHAPES 3
END

```

*A Shape as a List of Numbers*

GETSH outputs a description of a shape in a particular slot. The description is a list of sixteen numbers, which describe the sixteen rows and eight columns of a shape as depicted in the shape editor. Each row can be described by a number from 0 to 255. If the number is 0, the row is blank. If the number is 255, all the columns in that row are filled in. Look at the picture on page 30 in the *Atari Logo Reference Manual*. It shows you how to decipher the numbers. For example, 129 means that the first column and last column of a row are filled in.

If you would like to see the description for the rectangle in slot 1, type:

```

PR GETSH 1
255 129 129 129 129 129 129 129 129
  129 129 129 129 129 129 129 255

```

An empty shape is described by a list of sixteen zeros.

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

The three variables that now contain the descriptions of the three rectangles can be listed by typing PONS.

```

MAKE "SHAPE1 [255 129 129 129 129 129 129 129
  129 129 129 129 129 129 129 255]
MAKE "SHAPE2 [0 0 0 255 129 129 129 129
  129 129 129 129 129 129 129 255]
MAKE "SHAPE3 [0 0 0 0 0 0 0 126 66 66 66 66 66 66 126]

```



### *Copying Shapes*

You can copy shapes from one slot into another. This is often useful when you want to make variations of a shape. For example, you can put the shape in slot 1 into slot 4 by typing

```
PUTSH 4 GETSH 1
```

Now using EDSH 4 you can change the copy of shape 1, which is now in slot 4, to make a new design.

---

## Sounds and Music

This section is intended to give the musically naive programmer insight into the relationship between sounds and music using T00T, which is the tone-generating command in Logo.

T00T takes four inputs. The first input is either 0 or 1 to indicate which of two Atari hardware voices makes the tone. The second input is the frequency or pitch of the tone in cycles per second (vibrations per second). The third input is the loudness or volume with a range of 0 through 15, and the fourth input is the duration in sixtieths of a second. A duration of 60 is one second; a duration of 15 is a quarter of a second. Try:

```
T00T 0 440 10 15
```

### *Frequency and Pitch*

The *frequency* of a sound is the number of oscillations it makes in a second. A large frequency number is a high sound. People can normally hear frequencies ranging from 15 to 20,000 per second and can sing frequencies of between 200 to 2000 vibrations per second.

*Pitch* is the name musicians give to frequency. Pitches are organized into *octaves*. Each octave is divided into twelve chromatic steps, and each pitch is *heard* as equidistant from its neighbor. Multiplying the frequency of a certain pitch by the twelfth root of 2 will give you the next pitch above it. The value of the twelfth root of 2, which is approximately 1.0595, works as the step size because two pitches are an octave apart if the frequency of one is twice the other and because there are twelve pitches to an octave.

### *Music Names and Pitch-Frequency Conversion*

In music, alphabetic names are used for the pitches. The letters A through G name seven pitches. The other five are referred to by one of these letters

---

By Greg Gargarian.



## SPECIAL FEATURES OF ATARI LOGO

followed by either a “#” (sharp) or “b” (flat) mark. “A#,” pronounced “A sharp,” means “the chromatic step *up* from A.” Likewise, “Ab,” or “A flat,” means “the chromatic step *down* from A.”

The following conversion chart shows the relationship between pitch names and frequencies. Each column of frequencies represents one octave of chromatic pitches. The OCTAVE procedure generates the frequencies for these pitches.

Pitch Names	Frequencies		
A	220	440	880
A# (Bb)	233	466	932
B	246	493	987
C	261	523	1046
C# (Db)	277	554	1108
D	293	587	1174
D# (Eb)	311	622	1244
E	329	659	1318
F	349	698	1397
F# (Gb)	370	740	1480
G	392	784	1568
G# (Ab)	415	830	1661

← lower octaves higher octaves →

For example, the frequency 440 is the pitch A in a midrange octave. Twice 440 is 880, which is A in a higher octave. Frequency 220 is A in a lower octave. OCTAVE is a procedure that plays and prints the twelve chromatic frequencies of an octave in ascending order.

```
TO OCTAVE :LOWFREQ
OCTAVE1 :LOWFREQ :LOWFREQ * 2
END

TO OCTAVE1 :LOWFREQ :HIGHFREQ
IF :LOWFREQ > :HIGHFREQ [STOP]
TOUT 0 :LOWFREQ 15 30
PRINT INT :LOWFREQ
OCTAVE1 :LOWFREQ * 1.0595 :HIGHFREQ
END
```

Try:

```
OCTAVE 220
OCTAVE 440
OCTAVE 880
```

You might want to play with the procedures OCTAVE.UP and OCTAVE.DOWN to familiarize yourself with octaves.

```
TO OCTAVE.UP :FREQ
TOUT 0 :FREQ 15 60
PRINT :FREQ
TOUT 0 :FREQ*2 15 60
```

```

PRINT :FREQ*2
END

TO OCTAVE.DOWN :FREQ
TOOT 0 :FREQ 15 60
PRINT :FREQ
TOOT 0 :FREQ/2 15 60
PRINT :FREQ/2
END

```

Try:

```

OCTAVE.UP 220
OCTAVE.UP 261
OCTAVE.UP 493
OCTAVE.DOWN 1568
OCTAVE.DOWN 698

```

Listen to VICTORY in the Pacgame project for an example of using octaves.

### *Duration and Loudness*

A note has pitch, loudness, and duration. TOOT's third input controls loudness. CRESCENDO changes only the loudness of a repeating note. (In music terminology *crescendo* means "to get gradually louder.")

```

TO CRESCENDO :LOUDNESS
IF :LOUDNESS > 15 [STOP]
TOOT 0 440 :LOUDNESS 30
CRESCENDO :LOUDNESS + 1
END

```

To make it easier for you to hear changes in either duration or loudness, type

```
SETENV 0 1
```

Now try:

```
CRESCENDO 1
```

TOOT's fourth input is duration. SPEEDUP changes only the duration of a note; in this case, it shortens its duration at each recursive call.

```

TO SPEEDUP :DURATION
IF :DURATION < 10 [STOP]
TOOT 0 440 15 :DURATION
SPEEDUP :DURATION - 5
END

```

Try:

```
SPEEDUP 60
```

## SPECIAL FEATURES OF ATARI LOGO

SETENV is a Logo command that controls how abruptly a sound or a tone stops sounding by shaping the *decay* of the sound. SETENV's first input is the voice number. The second input determines whether the decay is abrupt (small numbers) or slow (larger numbers). Try CHANGE.DECAY to hear the effects of SETENV.

```
TO CHANGE.DECAY :DECAY
IF :DECAY > 10 [STOP]
SETENV 0 :DECAY
TOOT 0 440 15 40
WAIT 20
CHANGE.DECAY :DECAY+1
END
```

Type:

CHANGE.DECAY 1

To restore SETENV, type:

```
TO SCALE :FREQLIST
IF EMPTY? :FREQLIST [STOP]
TOOT 0 FIRST :FREQLIST 15 30
SCALE BUTFIRST :FREQLIST
END
```

*Scales and Intervals, Half Steps and Whole Steps*

A *scale* is a sequence of pitches within the limits of an octave. Scales are written in either ascending or descending order. Most Western scales have seven steps. The most complete scale is the *chromatic scale*, which contains all of the pitches in an octave.

The distance between pitches is called an *interval*. An octave is also an interval. The Ear Training project, found in this book, is an exercise in hearing intervals.

A *half step* is one chromatic step and a *whole step* is two chromatic steps. The half and whole steps are the building blocks of most traditional Western scales.

We can think of a scale as a stairway whose sequence of whole and half steps shape its ascent. For example, the C major scale is made of half and whole steps. The scale starts and ends on the pitch C. Try the following to hear the scale.

```
MAKE "C.MAJOR [261 293 329 349 392 440 493 523]
SCALE :C.MAJOR
```

*Melodies*

*Melodies* are invented sequences of *pitches* and *rhythms* (patterns of durations). Melodies can be any length and can combine with other melodies to



make songs. When we talk about melodies, we are talking about making music. The Melodies project deals extensively with constructing and playing with melodies.

### Noise Sounds

Hissing, sputtering, grunting, and other such sounds are *noise sounds*. When several frequencies are very close together, they can mix and become noise. The bouncing or siren sounds found in some of the projects in this book or in the Sound Effects project demonstrate that noise sounds can be not only expressive but quite beautiful as well.

---

#### PROGRAM LISTING

---

```

TO OCTAVE :LOWFREQ
OCTAVE1 :LOWFREQ :LOWFREQ * 2
END

TO OCTAVE1 :LOWFREQ :HIGHFREQ
IF :LOWFREQ > :HIGHFREQ [STOP]
TOOT 0 :LOWFREQ 15 30
PRINT INT :LOWFREQ
OCTAVE1 :LOWFREQ * 1.0595 :HIGHFREQ
END

TO OCTAVE.UP :FREQ
TOOT 0 :FREQ 15 60
PRINT :FREQ
TOOT 0 :FREQ*2 15 60
PRINT :FREQ*2
END

TO OCTAVE.DOWN :FREQ
TOOT 0 :FREQ 15 60
PRINT :FREQ
TOOT 0 :FREQ/2 15 60
PRINT :FREQ/2
END

TO SPEEDUP :DURATION
IF :DURATION < 10 [STOP]
TOOT 0 440 15 :DURATION
SPEEDUP :DURATION - 5
END

TO CRESCENDO :LOUDNESS
IF :LOUDNESS > 15 [STOP]
TOOT 0 440 :LOUDNESS 30
CRESCENDO :LOUDNESS + 1
END

TO CHANGE.DECAY :DECAY
IF :DECAY > 10 [STOP]
SETENV 0 :DECAY
TOOT 0 440 15 40
WAIT 20
CHANGE.DECAY :DECAY+1
END

TO SCALE :FREQLIST
IF EMPTY :FREQLIST [STOP]
TOOT 0 FIRST :FREQLIST 15 30
SCALE BUTFIRST :FREQLIST
END

MAKE "C.MAJOR [261 293 329 349 392 ►
440 493 523]

```

---

## Demons, Turtle Collisions, and Other Events

### *Demons and Events*

Atari Logo provides you with ways to detect certain events, such as whether a joystick is changing position or whether one turtle is colliding with another. You can write programs to watch for these events or you can set up special Logo helpers called *demons*. Demons are invisible creatures and run independently of your programs. You can create a demon on command and also get rid of it on command.

Many of the projects described in this book take advantage of Atari Logo's event detection. In this section we give examples of the different kinds of events and ways of checking for them.

### *Collisions*

Two kinds of collisions can be detected: a turtle colliding with another turtle or a turtle colliding with a line drawn on the screen.

#### Collisions Between Turtles

Logo can detect collisions between any two of the four turtles. There are six possible ways in which the four turtles can collide. Each of these possibilities is represented by an *event code* number. It is not necessary for you to learn which number represents which event since you can use the operation `TOUCHING`.

For example, you can type:

```
PR COND TOUCHING 0 1
```

and if turtle 0 and turtle 1 are touching one another, Logo responds `TRUE`; otherwise Logo responds `FALSE`. If you type:

```
PR TOUCHING 0 1
```

Logo prints the event code, which in this case is 19.

`TOUCHING` expects as its two inputs any of the turtle numbers from 0 through 3. `TOUCHING` outputs the event code, which is a number from 16 through 21.

`COND` is the Logo operation that outputs `TRUE` if an event is occurring and `FALSE` if the event is not occurring at the time `COND` checks. Sometimes the event is not noticed since it is not occurring when `COND` checks. In the next example, `COND` is used in a recursive procedure that continually peeks at the condition of the events.

**Using COND**

Try the following:

```
TO FLOAT
  TELL [0 1 2 3] CS
  EACH [SETH WHO * 90]
  SETSP 30
  TOUCHTOOT
END

TO TOUCHTOOT
  IF COND TOUCHING 0 2 [TOOT 0 220 5 15]
  IF COND TOUCHING 1 3 [TOOT 0 880 5 15]
  TOUCHTOOT
END
```

TOUCHTOOT looks for collisions between two sets of turtles. Whenever a collision is detected, a sound is emitted. In one case, the sound is a low A (220). In the other case, the sound is a high A (880).

Try:

**FLOAT**

FLOAT sets up the turtles and then turns the job of detecting the collision over to TOUCHTOOT. As long as the program is running, Logo emits a low A (220) when turtles 0 and 2 touch and a high A (880) when turtles 1 and 3 touch.

**Using Demons**

You can create a demon by using the command WHEN. For example:

```
WHEN TOUCHING 0 2 [TOOT 0 220 5 30]
```

You could change FLOAT so that it sets up demons:

```
TO FLOAT
  TELL [0 1 2 3] CS
  EACH [SETH WHO * 90]
  SETSP 30
  WHEN TOUCHING 0 2 [TOOT 0 220 5 15]
  WHEN TOUCHING 1 3 [TOOT 0 880 5 15]
END
```

FLOAT sets up the turtles and two demons. Try:

**FLOAT**

You can see how many demons are active by typing

**PODS**

Logo responds by typing the WHEN instructions with the event code numbers:



## SPECIAL FEATURES OF ATARI LOGO

```
WHEN 17 [T00T 0 880 5 15]
WHEN 20 [T00T 0 220 5 15]
```

Thus, even if you do not directly refer to the event code, Logo does. You can always look the codes up in the reference manual when you need them; you can derive the collision event codes experimentally by typing, for example:

```
PR TOUCHING 0 3
```

The demons remain active until you type CS, shut the machine off, or explicitly get rid of the demons.

## Collisions Between a Turtle and a Line

Each turtle can collide with lines drawn by any of the three pens. The operation OVER can be used to describe an event in much the same way as TOUCHING does. OVER takes two inputs: the first is the turtle number and the second is the pen number. OVER outputs an event code number.

There are twelve possible collisions. If you want to know a code number, you can always type something like the following:

```
PR OVER 0 0
```

Logo responds:

```
0
```

```
PR OVER 3 2
```

```
14
```

Notice that both TOUCHING and OVER output numbers. They are not predicates and do *not* output TRUE or FALSE.

Here is an example that uses this kind of collision. We make a square on the screen with pen 1. Then turtle 0 is put in the square and given a speed.

```
TO DRAWSQUARE
CS HT TELL 0 ST SETPN 1 PU
SETPOS [-50 -50] PD
REPEAT 4 [FD 100 RT 90]
PU
HOME
END
```

Next we make a procedure that runs DRAWSQUARE, puts the turtle in motion, and sets up a demon to look for turtle 0 colliding with lines drawn by pen 1.

```
TO BOXIT
DRAWSQUARE
WHEN OVER 0 1 [RT 180]
SETSP 20
END
```

The turtle monotonously bumps into a side of the square, turns around, and travels until it again bumps into a side of the square and repeats the process. You might want to change the amount the turtle turns. For example, try  
RT 160.

### *Detecting the Tick of a Clock*

Another detectable event is the passing of a particular unit of time, which in Atari Logo is a second. Checking for this event in a program is not very meaningful since more than a second may pass before the instruction using COND is invoked. This event calls for a demon.

To set up a once-per-second demon, you use the Logo command WHEN. For example:

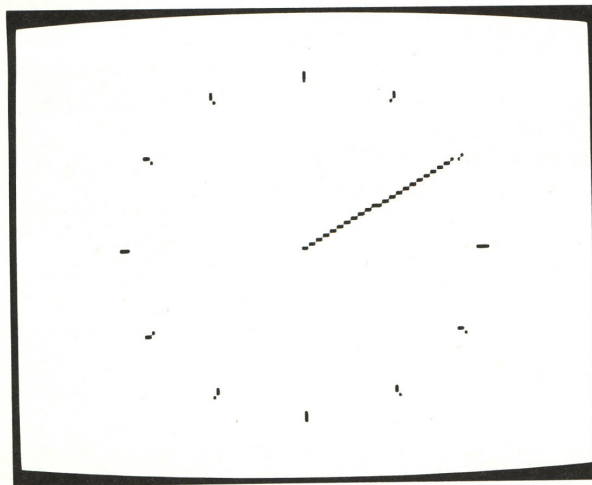
```
TELL [0 1 2 3] CS PD ST
EACH [SETH WHO * 90]
WHEN 7 [FD 15 RT 15]
```

A demon is created and told to watch for event 7 (the passing of a second). WHEN's second input is a list of Logo instructions. The demon has Logo run this list of instructions when it detects the event it is watching for.

In this example the four turtles are set up to head in different directions. Then the demon has Logo run the two instructions FD 15 and RT 15 every second.

Here is another example that uses a demon to watch for event 7 (the clock tick). In this example, turtle 0 makes a real clock.

```
TO TURTLE.TIMER
CS TELL 0 FS PU
REPEAT 12 [FD 100 PD FD 5 PU BK 105 RT 30]
PX
WHEN 7 [FD 95 WAIT 45 BK 95 RT 6]
END
```





*Getting Rid of Demons*

You can dismiss a particular demon by retyping the `WHEN` command, but this time with an empty instruction list. For example, to get rid of the demon used in `TURTLE.TIMER`, type:

```
WHEN 7 []
```

The clock will remain on the screen but the hand will no longer be visible.

You dismiss all demons whenever you type `CS`.

Logo will dismiss all demons if an error occurs in a demon's list of instructions.

*Using a Joystick*

Two events you can detect have to do with joysticks, in one case detecting whether any joystick button has been pressed and in the other case detecting whether the joystick itself has changed position. For example, you might have the turtles moving about the screen using `FLOAT`. You might want to change the turtles' direction by pressing a joystick button. `CHECKJOY` will let you do that; it will make a sound whenever you change the position of a joystick.

```
TO CHECKJOY
WHEN 3 [RT 180]
WHEN 15 [T00T 0 440 5 30]
END
```

You might want to edit `FLOAT` so that it calls `CHECKJOY` after setting up the turtles (as its last instruction).

Event 3 occurs when any joystick button is pressed. Event 15 occurs when any joystick changes position.

*Advice*

- You cannot create a demon when you first start Logo or immediately after you leave the Logo editor. Before the demons will function, you must use a graphics command such as `CS`. If you want to use a once-per-second demon to print on the text screen, you might do the following:

```
CS TS
WHEN 7 [PR "HOORAY]
```

- Collisions are detected only when the turtle involved is not hidden.
- Many projects in this book use demons to watch for turtles colliding with lines. In many cases the events will not be detected in time; the turtles escape and other measures have to be taken. These escapes take place when the turtles are moving at a fast rate of speed or when there are many demons at work.
- If you are looking for a collision that takes place while the turtles are in motion, then demons are most appropriate. Demons are also appropriate if you are looking at constantly changing events. Most of the projects in this book use demons. *Cartoon* (p. 98) and *Four-Corner Problem* make use of `COND`.



# Name and Subject Index

- Abelson, Harold, 115, 212, 303  
 Adding Numbers, 258–266  
   ADD, 266  
 Addition table:  
   in Adding Numbers, 260  
   in preface, vii  
 Adventure, 172–191  
   ADVENTURE, 188  
 Alien, 160–171  
   START, 167  
 Animal Game, 11–21  
   ANIMALGAME, 20  
 Animating Line Drawings, 227–229  
   START, 229  
 Animation:  
   by changing shapes: in Blaster, 152  
     in Cartoon, 87ff.  
     in Exercise, 80 ff.  
     in Jack and Jill, 108ff.  
     in Pacgame, 141  
     in Rocket, 126  
     in Turtles and Their Shapes, 367–368  
   colormap, 227  
   of human figures (*see* Alien; Exercise;  
     Jack and Jill; Rocket)  
   of line drawings (*see* Animating Line  
     Drawings)  
   using invisible walls, 93  
 Arctan, 212–213  
   ARCTAN, 213  
 Argue, 6–11  
   ARGUE, 11  
 Arithmetic:  
   in Adding Numbers, 258–266  
   in Math: A Sentence Generator, 39–45  
     MATH, 45  
   in Number Speller, 46–50  
 Arithmetic operations, 39, 258  
 Aspect ratio, 269, 283
- Behensky, Max, 242, 251  
 Bestline, 337–347  
   BESTLINE, 346  
 Binary digits (bits), x, 283  
 Biology:  
   in Bestline, 337  
   in preface, x
- Bits (binary digits), x, 283  
 Blaster, 151–159  
   BLASTER, 158  
 Boxgame, 133–140  
   BOXGAME, 138  
   BYEBYE, 139  
   JOYGAME, 139  
   NEWGAME, 139  
 Bytes, 283
- Cartoon, 87–104  
   CARTOON, 102  
 Cassette, 282  
 Chandler, Jeanry, 160, 191  
 Cipher, 330  
 Clock, 379  
 Collisions:  
   COND, 376, 377  
   Demons, 377–378  
   OVER, 378  
   TOUCHING, 376  
   WHEN, 377–378  
 Color:  
   background, 365–366  
   pen, 363–365  
   turtle, 362–363  
 Commands, 272, 326  
 Computer literacy, vii  
 COND, 376, 377  
 Cotten, Susan, 21, 74, 80, 126
- Data base, 8  
 Data representation, 282, 335  
   (*See also* List, as data representa-  
   tion)  
 Data structure, 335  
 Davis, James, 172, 302  
 Decay of sound, 374  
 Delpit, Lisa, 35  
 Demon joystick, 376, 380  
   (*See also* Joystick)  
 Demons, 376  
   in Alien, 162, 166  
   in Blaster, 153, 155  
   in Boxgame, 133–134  
   in Cartoon, 93–96
- Demons (*Cont.*):  
   in Demons, Turtle, Collisions, and Oth-  
     er Events, 376–380  
   in Dungeon, 193  
   in Jack and Jill, 111–112  
   in Lines and Mirrors, 356  
   in Pacgame, 140–142, 147  
   in Rocket, 126, 127  
 Dictionary, 21–27  
   DICTIONARY, 26  
 DiSessa, Andrea, 115  
 Diskette, Logo files for all projects, xvi  
   (*See also* Files)  
 Display Workspace Manager (DWM),  
   292–301  
   DWM, 296–301  
 Drawing Letters, 50–58  
   LETTER, 57  
   SAY, 57  
 Dula, Annette, 58  
 Dungeon, 191–205  
   START, 198  
 Duration, 230, 373  
 DWM (Display Workspace Manager),  
   292–301
- EACH, 207  
 Ear Training, 239–242  
   EAR.TRAINING, 241  
   (*See also* Sounds and Music)  
 Editor, shape, 366  
 Electronic mail (*see* Mail)  
 English sentences, 1, 172  
   (*See also* Sentence generator, English)  
 Error, minimum, 340  
 Event code, 376  
 Events, 376  
 Exercise, 80–87  
   EXERCISE, 85
- Files:  
   loading additional procedures from:  
     in Dungeon, 197  
     in Jack and Jill, 117  
   storing data in: in Display Workspace  
     Manager, 294, 296

- Files, storing data in (*Cont.*):  
   in Mail, 58, 65  
   in Savepict and Loadpict, 282, 284  
 Fill (filling drawings with color), 267–282  
   FILL, 281  
 Filled-in POLY, 218–222  
 Four-Corner Problem, 210–211  
   FOUR, 211  
 Frequency, 371
- Games (*see* Adventure; Alien; Animal Game; Argue; Blaster; Boxgame; Dungeon; Hangman; Madlibs; Pacgame; Wordscram)
- Gargarian, Gregory, 230, 239, 242, 371  
 Global variable, 17  
 Gongram: Making Complex Polygon Designs, 214–222  
   GONGRAM, 222  
 Grammar, 1  
 Grandfield, Michael 104, 140, 336  
 Graphics screen, 267–291  
 Greenberg, Bernard, 11
- Hangman, 27–38  
   HANGMAN, 36  
 Hardebeck, Edward, ix, 302  
 Harvey, Brian, 27, 46, 50, 74, 151, 206, 210, 212, 267, 282, 292, 322, 331, 343  
 Hillis, Danny, ix, 6, 331
- Indirect reference (*see* Naming)  
 Integer addition, 258–266  
 Interpreter, Logo, xi, 302–322  
   arglist, 306  
   args, 306  
   arguments (args), 306  
   bind frame, 313, 315–316  
   binding, 313–314  
   EVAL, 304  
   evaluation, 304  
   evaluator, 304  
   INIT, 321  
   inputs to procedures, 306, 312  
   LOGO, 319  
   primitive, 302, 307  
   programming language, 302, 303  
   quoting, 305  
   stack, 314–315  
   value, 303, 305  
   variables, binding, 313  
 Interval, musical, 239, 374  
 Item (*see* List)  
 ITEM, 38, 50, 191  
 Iteration, 329
- Jack and Jill, 104–124  
   JACK, 117
- Joystick:  
   in Alien, 162, 166  
   in Blaster, 151–152  
   in Boxgame, 137–138  
   in Demons, Turtle, Collisions, and Other Events, 376, 380  
   in Dungeon, 192, 198, 200
- Knuth, Donald, 336
- Least squares, 338  
 Letter in word (*see* Membership of letter in word)  
 Linear regression, 338  
 Lines and Mirrors, 347–361  
   BOUNCE, 360  
 LISP, programming language, 173  
 List:  
   choose item from, 2, 23, 28, 36, 47, 63–64, 69, 76–78, 242  
   as data representation: in Adventure, 79  
   in Bestline, 342  
   in Display Workspace Manager, 295  
   in Dictionary, 21  
   in Hangman, 28  
   in Lines and Mirrors, 352–357  
   in A Logo Interpreter, 311  
   in Madlibs, 76  
   in Mail, 60  
   in Map, 322–331  
   in Mergesort, 331–337  
   in Naming Notes, 255  
   in Savepict and Loadpict, 286  
   in Wordscram, 74  
   flat, 330–331  
   iteration over, 329  
   mapping over, 322, 343  
   pick random item from, 2, 38  
   property, 173  
   reduction, 328–331  
   BIGE, 343  
   representing program: in Drawing Letters, 52  
   in Jack and Jill, 107  
   in A Logo Interpreter, 311  
   representing shapes, 370  
   representing stack, 314–315  
   representing tree, 13  
   sorting, 295, 331  
 Little person model, 260, 262  
 Loading shapes, 144, 153, 369  
 Loadpict, Savepict and, 282–291  
   LOADPICT, 291  
 Local variable, 272  
 Logo Interpreter (*see* Interpreter, Logo)  
 Loudness, 230, 374
- Madlibs, 74–79  
   MADLIB, 78
- Mail, 58–67  
   MAIL, 65  
 MAKE with nonquoted first input, 9, 252  
 Map, 322–331  
   MAP, 331  
   MAP.LIST, 331  
   MAP.WORD, 331  
   REDUCE, 331  
 Mapping, 323–325  
 Math (*see* Arithmetic)  
 Melodies, 230–239  
   SEQUENCER, 238  
   SONG, 238  
   (*See also* Sounds and Music)  
 Membership of letter in word:  
   in Hangman, 31  
   in Map, 327  
   in Wordscram, 68  
 Menu, 295  
 Mergesort, 331–337  
   SORT, 337  
 Minsky, Henry, 302  
 Minsky, Julie, 337  
 Minsky, Margaret, 6, 160, 191, 242, 251  
 Minsky, Marvin, vii  
 Mintz, Toby, 347  
 Mirrors (*see* Lines and Mirrors)  
 Mountains (*see* Cartoon; Jack and Jill; Rocket)  
 Music (*see* Sounds and Music)
- NAMEP, 9  
 Naming, 8, 146, 251  
 Naming Notes, 251–257  
   MUSIC, 256  
   NAMENOTES, 256  
 Note, musical (*see* Sounds and Music, note)  
 Number Speller, 46–50  
   SPELL, 50
- Operations, 272, 326  
 Optics, in Lines and Mirrors, 347  
 OVER, 378
- Pacgame, 140–151  
   PACGAME, 149  
 Papert, Seymour, viii  
 Pens, 363–365  
 Picture files on diskette (*see* Files, storing data in)  
 Pitch, 371, 373  
 Pixels (screen dots), 269, 283  
 Plotting graphs, 337ff.  
 Polycirc, 222–227  
   POLYCIRC, 227  
 Polygon (*see* Animating Line Drawings; Gongram: Making Complex Polygon Designs; Polycirc)



Positioning text on screen, 30–31, 292  
 Predicate, 173, 181  
 Primitive, 266, 302, 307  
 Programming languages, xi, 302ff.  
 Property lists, 173

Quoting special characters, 7, 42

RC, 22, 61, 299  
 Recursion, 258, 275, 306, 333  
 Reflecting shapes, 157  
 Rhythm, 236, 374  
 Rocket, 124–132  
     ROCKET, 128  
 Rotating shapes, 369  
 RUN, 179, 204, 308, 324  
 Run-length encoding, 290

Savepict and Loadpict, 282–291  
     LOADPICT, 291  
     SAVEPICT, 290, 291  
 Saving pictures, 282  
 Saving shapes, 144, 157, 369  
 Scale, musical, 374  
 Screen dots (pixels), 269, 283  
 Sengen: A Sentence Generator, 1–6  
     SENGEN, 5–6  
 Sentence generator:  
     English, 1–6  
         SENGEN, 5  
     math, 39–45  
         MATH, 45  
 SETENV, 249, 374  
 Seven-segment digits, 50  
     (See also Drawing Letters)  
 Shape editor, 366  
 Shape lists, 370  
 Shapes, 366–371  
     loading, 144, 153, 369  
     reflecting, 157  
     saving, 144, 157, 369  
 Sharman, Keith, 67  
 Simulation, viii, x, 172

Slope, 340, 348  
 Societies, computer programs as, x  
 Solomon, Cynthia, 1, 39, 133, 362, 366,  
     376  
 Solomon, Erric, 87, 214, 222, 227  
 Sorting, 295, 331–337  
 Sounds and Music:  
     chromatic step, 372  
     decay, 374  
     duration, 230, 373  
     frequency, 371  
     glissandi, 244  
     half step, 374  
     interval, 239, 374–375  
     loudness, 230, 374–375  
     note, 231–242, 251–257, 371–373  
         names, 231, 251, 371  
     octave, 371  
     pitch, 371, 373  
     rest, 231  
     rhythm, 236, 374  
     scale, 374  
     sequencer, 236  
     SETENV, 249, 374  
     TOOT, 371  
     transpose, 234  
     trill, 247  
     whole step, 374  
 Sound Effects, 242–250  
     advance and retreat, 243–244  
     ambulance siren, 242, 245  
     bird, 97, 117, 248  
     boing, 246  
     bounce, 249  
     echo, 250  
     fanfare, 248–249  
     glissandi, 244–245  
     motorcycle, 245  
     rocketship, 126  
     siren, 242, 245  
     sliding, 244–245  
     Space Invader game, 160  
     spaceship, 245  
     trill, 247  
 Sparse data representation, 286  
 Speed, turtle, 362  
 Spelling numbers (see Number Speller)

Stop rule, 48  
 Subtree, 13  
 Sussman, Gerald J., 303  
 Szneech, Martian, x

Teaching, 258ff.  
 Template, 53, 325  
 THING, 8, 146, 251  
 Tinkertoy tic-tac-toe machine, ix  
 TOOT, 371  
 TOUCHING, 376  
 Towards and Arctan, 212–213  
     ARCTAN, 213  
     TOWARDS, 213  
 Trees (diagram):  
     data representation, 12–13  
     examining, 14  
     represented as list, 13  
 Trees (drawing), 115  
 Turing, Alan, xi  
 Turtle Collisions, 376–379  
 Turtle Graphics, 362–366  
 Turtle Race, 206–210  
     RACE, 210  
 Turtles and Their Shapes, 366–  
     371

Variables:  
     as data base, 8, 251, 309  
     global, 17  
     local, 272

Weinreb, William, 11  
 WHEN, 377–380  
 WHO, 362  
 Word, iteration over (mapping over),  
     327–328  
 Wordscram (word scrambling game), 67–  
     74  
     WORDSCRAM, 71  
 Workspace management (see Files)

Young, Lauren, 124

---

# Procedures Index

---

- ?, 189
- %CS, 321
- %FD, 321
- %IF, 321
- %MAKE, 321
- %PRINT, 321
- %RT, 321
- %TO, 321
  
- A, 129
- ABS, 213, 246, 361
- ABSENT, 189
- ACTIVATE, 121
- ADD, 45, 204, 266
- ADD1, 266
- ADD.ENTRY, 26
- ADD.ENTRY1, 26
- ADDGUESS, 73
- ADDIT, 266
- ADD.TO.SCORE, 150
- ADDUP, 266
- ADJECTIVE, 6
- ADJECTIVES, 6
- ADJUST, 168
- ADVANCE, 243
- ADVANCE1, 243
- ADVANCE2, 243
- ADVENTURE, 188
- AFTERMATH, 120
- AGITATION, 247
- ALREADY, 37
- ALTER, 20
- AMOVE, 168
- ANIMALGAME, 20
- ANIMATE, 121
- ANIMATE.PACMAN, 150
- ANOTHER?, 74
- ANOTHER.HEARING, 242
- ANSWER, 45
- ANSWER1, 45
- ANSWER2, 45
- A.OR.AN, 20
- APPLY, 320
- APPLY.SFUN, 320
- APPLY.UFUN, 320
- ARCR, 85
  
- ARCTAN, 213, 361
- ARCTAN.RAD, 213, 361
- ARGUE, 11
- ARGUEWITH, 11
- ARM, 37
- ARMOR, 202
- ARROW, 57
- ARTICLE, 79
- ASETUP, 168
- ASSIGNKEYS, 256
  
- B, 139, 360
- BACKGROUND.SCENE, 130
- BACK.LEG, 86
- BADTRY, 37
- BAY, 121
- BEAT, 132
- BECOME.BIRDS, 121
- BESTLINE, 346
- BETWEEN, 361
- BIGE, 346
- BIND.ARG, 321
- BIND.ARGs, 321
- BIND.FRAME, 321
- BIRD, 102
- BIRDS, 248
- BIRDSONG, 248
- BIRDSONG1, 248
- BIRDS.SIGN, 102
- BLADE, 168
- BLASTER, 158
- BLASTER.LOOP, 159
- BLAST.OFF, 130
- BLAST.OFF2, 130
- BLAST.OFF.DOWN, 130
- BLAST.SOUND, 131
- BOING, 246
- BOING1, 246
- BOMB, 168
- BOMB.DEMONS, 168
- BONGO, 239
- BORDER, 360
- BOUNCE, 249, 360
- BOUNCE1, 249
- BOUND?, 320
- BOX, 138
  
- BOXGAME, 138
- BOXIT, 378
- BRAG, 20
- BRANCH, 118
- BUILD, 121
- BUILD.HOUSE, 120
- BULLSEYE, 150
- BUTLAST3, 50
- BYEBYE, 139
- BYTEPOS, 282
  
- C, 120
- CANCEL.DEMONS, 169
- CARRYINGP, 189
- CARTOON, 102
- CHANGE.DECAY, 375
- CHECK, 361
- CHECK.ANSWER, 241
- CHECKLIST, 242
- CHECK.SLANT, 361
- CHECK.VERT, 361
- CHEST, 201
- CLEANUP, 321
- CLEAN.UP, 132
- CLEARMESSAGE, 37
- CLIMB, 238
- CLIMBING, 238
- COLLAPSE, 86
- COLLATE.BEFORE, 337
- COLLATE.BEFORE.WORD, 337
- COLOR.AT, 281
- COMPARE, 73, 337
- COMPARE2, 337
- COMPARE3, 337
- CONNECT, 5
- CONNECTS, 5
- CONVERSATION, 132
- CRESCENDO, 375
- CYC, 229
- CYCLE, 229
  
- D, 203, 360
- DEC.SOUND, 139
- DEF.SFUN, 321
- DEF.SSFUN, 321
- DEF.UFUN, 321



- DELETE, 67
- DEPARTURE, 245
- DESCRIBE, 188
- DICTIONARY, 26
- DIE, 159
- DIGIT, 50
- DIMENSIONS, 198
- DISAPPEAR, 103
- DISK.DUMP, 67
- DISPLAY, 37
- DISWORD, 37
- DIV2, 45
- DIVIDE, 45
- DMOVE, 168
- DO.CHOICE, 26
- DO.DOTS, 73
- DOOR, 121
- DOOR.BELL, 121
- DOOR.LINE, 199
- DOORP, 188
- DOOR.SIDE, 199
- DOT, 281
- DOT0, 281
- DOT1, 281
- DOT2, 281
- DOT3, 281
- DOTA, 281
- DOTTED?, 320
- DOWN, 245
- DOWN.THE.HILL?, 121
- DRAW, 57, 361
- DRAW1, 38
- DRAW2, 38
- DRAW3, 37
- DRAW4, 37
- DRAW5, 37
- DRAW6, 37
- DRAW7, 37
- DRAW.BOARD, 149
- DRAWBOX, 139
- DRAWBOX.BYE, 139
- DRAW.FINISHLINE, 210
- DRAWGRASS, 102
- DRAW.LINES, 361
- DRAW.MARS, 128
- DRAW.RACETRACK, 210
- DRAWSQUARE, 378
- DRAW.TRAMPOLINE, 85
- DRINK, 190
- DROP, 189
- DROPANDLAND, 102
- DROP.TURD, 102
- DSETUP, 168
- DWM, 296
- DWM.1, 296
- DWM.BEFORE, 297
- DWM.CLEAR.PROMPT, 301
- DWM.CMD.2, 299
- DWM.CMD.21, 299
- DWM.DISKSAVE, 301
- DWM.DOWN, 299
- DWM.DRAW.M1, 298
- DWM.DRAW.MENU, 297
- DWM.EDIT, 300
- DWM.ERASE, 301
- DWM.ERASE1, 301
- DWM.FIRSTPART, 297
- DWM.FLUSH.MARKS, 300
- DWM.HIDE.CURSOR, 299
- DWM.IGNORE, 300
- DWM.INSERT, 297
- DWM.ITEM, 298
- DWM.LEFT, 299
- DWM.MAIN.LOOP, 298
- DWM.MARK, 300
- DWM.MARKLIST, 300
- DWM.MARKLIST1, 300
- DWM.PRINTER, 301
- DWM.PRINTOUT, 300
- DWM.PROCLIST, 296
- DWM.PROMPT, 299
- DWM.READ.TITLE, 297
- DWM.READ.TITLE1, 297
- DWM.REMOVE, 300
- DWM.RIGHT, 299
- DWM.SAVE, 301
- DWM.SET.CURSOR, 299
- DWM.SHORT, 298
- DWM.SHORT1, 298
- DWM.SHOW.CURSOR, 299
- DWM.SIZE.MENU, 297
- DWM.SIZE.TABS, 297
- DWM.STAR, 298
- DWM.TOGGLE.MARK, 300
- DWM.UNMARK, 300
- DWM.UP, 299
- DX, 360
- DY, 360
- EAR.TRAINING, 241
- EAR.TRAINING2, 241
- ECHO, 250
- EDGE, 281
- ENDING, 250
- ENDING1, 250
- ENTER.TRUCK, 120
- ERASE.POLY, 222
- ERASE.POLY.FILL, 222
- EVAL, 319
- EVAL.ARGS, 320
- EVAL.BODY, 321
- EVAL.BODY1, 321
- EVAL.CALL, 320
- EVLINE, 319
- EVLINE1, 319
- EXAMINE, 189
- EXERCISE, 85
- EXPLODE, 150, 159
- EXPLODE1, 150
- EXPLODE.AW.D, 170
- EXPLODE.B.AW, 170
- EXPLODE.BOMB, 169
- EXPLODE.BOMB.D, 169
- EXPLODE.BOMB.MISSILE, 169
- EXPLODE.BULLET.AW, 170
- EXPLODE.MISSILE.A, 169
- EXPLORE, 20
- EYES, 38
- F, 139
- FANFARE, 150, 249
- FANFARE1, 150, 249
- FASTBOUNCE, 250
- FIGB, 360
- FIGH, 360
- FIGHT, 200
- FIGLINE, 360
- FIGM, 360
- FIGTURN, 361
- FILL, 281
- FILL1, 281
- FILL2, 281
- FILL.BLANK, 78
- FILL.BOTH, 281
- FILL.CHUNK, 282
- FILL.IN, 78
- FILL.LINE, 281
- FILL.RAY, 282
- FILL.UP, 281
- FILL.UPI, 281
- FIND.BG, 282
- FINISH, 38, 57
- FINISH.UP, 20
- FIRE, 159, 169
- FIRST.CLEANUP, 120
- FIRST.N, 336
- FIRST.PART, 336
- FIXGOT, 37
- FLAP, 103
- FLAPFLAP, 103
- FLASH, 140
- FLASH.COLOR, 131
- FLIGHT.SOUND, 132
- FLIP, 103, 362
- FLOAT, 377
- FLY, 103, 121
- FLYOFF, 103
- FORMAT, 27, 337
- FOUR, 211
- FOUR.LOOP, 211
- FRONT.LEG, 86
- FROWN, 38
- FSET, 320
- FSYM, 320
- FSYMEVAL, 320
- GALL1, 36
- GALLOWS, 36
- GAMELOOP, 188
- GATHER.ARGS, 321

- GE, 361
- GET, 72
- GETANSWER, 71
- GETDOOR, 190
- GET.ENTRY, 26
- GETGUESS, 36, 72
- GETINP, 45
- GETINPUT, 188
- GETLINE, 360
- GET.LINE, 322
- GET.MESSAGE, 66
- GET.NEW.QUESTION, 20
- GET.OFF.TRAMPOLINE, 86
- GET.PILOT, 130
- GET.RIGHT.GUESS, 20
- GETTURN, 139
- GET.VARIABLE.VALUE, 320
- GETWORDS, 74
- GOLD, 201
- GONGRAM, 222
- GOODBYE, 131
- GOROOM, 188
- GPROP, 191
- GPROPI, 191
- GRASS, 102, 118
- GREATEST.NUM, 347
- GREEN.PLANET, 131
- GROW, 368
- GUESS, 20
  
- HANGMAN, 36
- HATCH, 103
- HELP, 66
- HILL, 118
- HILL1, 118
- HINT, 73
- HINTWORD1, 73
- HIT.GREEN.PLANET, 131
- HIT.SOUND, 131
- HOME.AGAIN, 128
- HPOTION, 203
  
- I, 203
- IGNORE, 322, 346
- INC.SOUND, 139
- INFOR, 360
- INIT, 188, 321
- INITOBS, 190
- INITPRIMS, 322
- INITROOMS, 190
- INITVARS, 190
- INPUT.LINE, 322
- INPUT.LINE1, 322
- INROOMP, 189
- INSTRUCTIONS, 71
- INTER1, 188
- INTER2, 188
- INTERPRET, 188
- INTERVALS, 242
- INVADE, 167
  
- INVENTORY, 189
- ITEM, 38, 50, 73, 191, 242, 360
  
- JACK, 117
- JACK.AND.JILL, 120
- JEWEL, 202
- JOYGAME, 139
- JOYRD, 139
- JOYREAD, 139
- JOYREAD.NEW, 140
- JUMP, 86, 256
- JUMPING.JACKS, 86
- JUMPING.JACKS1, 86
- JUMPING.ON.TRAMPOLINE, 86
- JUMP.OVER.RT, 121
- JUSTX, 346
- JUSTY, 346
  
- KEEPSHAPES, 370
- KEY, 201
- KEYBOARD, 257
  
- L, 139
- LAND, 169
- LANDING, 103
- LAST3, 50
- LAST.N, 336
- LAST.PART, 336
- LBRANCH, 118
- LEARN, 20
- LEARN.LINES, 360
- LEARNOPP, 11
- LEAST.NUM, 347
- LEAST.SQUARES.SLOPE, 346
- LEG, 364
- LETTER, 57
- LEVEL, 202
- LINE.EQUATION, 346
- LOADPICT, 291
- LOADPICT1, 291
- LOAD.SHAPES, 118
- LOGO, 319
- LOGOLOOP, 319
- LOOK, 188
- LOOP, 361
- LOSE, 168
- LOVE.HATE, 11
  
- M, 128, 360
- MADLIB, 78
- MAIL, 65
- MAKE.ENTRY, 27
- MAKEROOM, 198
- MAKE.SFUN.DEF, 320
- MAKETREE, 20
- MAKE.UFUN.DEF, 320
- MAP, 331
- MAP.LIST, 331
  
- MAP.WORD, 331
- MARS, 128
- MATH, 45
- MATH1, 45
- MDEAD, 202
- MEMP, 37
- MERGE, 336
- MHIT, 202
- MISSILE, 168
- MISSILE.DEMONS, 168
- MOON.MARS, 131
- MOON.PLANET, 131
- MORE.STARS, 131
- MOTORCYCLE, 245
- MOUNTAIN, 102
- MOUNTAINS, 130
- MOUNTAINS.DISAPPEAR, 131
- MOUTH, 38
- MOVE, 188, 204
- MOVE1, 188, 210
- MOVEPLAYER, 200
- MOVE.TO.TRAMPOLINE, 86
- MOVEWING, 103
- MULTIPLY, 45
- MUSIC, 256
- MUSIC.MIRROR, 238
- MY.MESSAGES, 66
  
- NAMENOTES, 256
- NAMEOCTAVE, 256
- NAMEOCTAVES, 256
- NARGS, 320
- NEWGAME, 139
- NEWHEAD, 361
- NEXT, 267
- NEXT.ITEM, 319
- NEXT.ITEM1, 319
- NEXTPEN, 227, 229
- NO, 151
- NO.BRANCH, 20
- NODE, 118
- NOISE, 131
- NOTE, 257
- NOUN, 6
- NOUNPHRASE, 5
- NOUNS, 6
  
- OCTAVE, 375
- OCTAVE1, 375
- OCTAVE.DOWN, 375
- OCTAVE.UP, 375
- ON.WITH.THE.GAME, 150
- OPPOSITE, 11
  
- PACGAME, 149
- PAIRS, 347
- PAIRUP, 347
- PDEAD, 202
- PHIT, 202
- PICK, 6, 38
- PICKWORD, 37



- PICTLOC, 291
- PITCH, 238, 242
- PITCH1, 238, 242
- PIXEL, 281
- PLANET.MARS, 128
- PLANET.SATURN, 129
- PLAY, 20, 36, 149
- PLAY.BLASTER, 158
- PLAYGAME, 71
- PLAYGAME1, 72
- PLAYTUNE, 256
- PLOT.AXES, 346
- PLOT.HORIZ, 347
- PLOTLINE, 346
- PLOT.POINTS, 346
- POINT, 168
- POINT1, 360
- POINT2, 360
- POLY, 227
- POLY1, 227
- POLYCIRC, 227
- POLY.FILL, 222
- POLYGON, 222
- POLYGON1, 222
- POP, 322
- POP1, 322
- POSADDR, 281
- POTION, 201
- PPOTION, 203
- PPROP, 191
- PPROP1, 191
- PRDOOR, 189
- PRDOORS, 189
- PRDOORS1, 189
- PRESENTP, 189
- PREXIT, 188
- PREXITS, 188
- PREXITS1, 188
- PRINT.ALL.MESSAGES, 67
- PRINT.AND.DELETE, 66
- PRINT.DEF1, 27
- PRINT.DEF2, 27
- PRINT.DEFINITION, 27
- PRINT.DICTIONARY, 27
- PRINT.ENTRY, 27
- PRINT.MESSAGE, 66
- PRINT.SCORE, 169
- PROPTRUE?, 191
- PUNCTUATE, 79
- PUSH, 322
- PUTBYTE, 291
- PUTSHAPES, 132, 158, 370
  
- QUESTION, 20
- QUOTED, 331
- QUOTED?, 319
- QUOTIFY, 320
- QUOTIFY1, 320
  
- R, 129, 139
- RACE, 210
- RACE.FROM, 210
- RAINBOW, 366
- RAINBOW1, 366
- RAMP, 122, 244
- RAMP2, 245
- RANDOM.POS, 159
- RANGE, 347
- RANPICK, 74
- RBRANCH, 118
- READ.BODY, 322
- READ.BODY1, 322
- READ.LINE, 322
- READY.JUMPING.JACKS, 86
- READY.SHAPES, 85
- RECEIVER'S.NAME, 66
- REDISPLAY, 74
- REDUCE, 331
- REFORM, 122
- REFORM.AND.EXIT.TRUCK, 120
- REFRESH.SCREEN, 72
- REMEMBER, 360
- REMOVE, 27, 72, 191
- REMOVE.ALL, 65
- REMOVE.ENTRY, 27
- RERUN, 122
- RESET, 122, 363
- RESETPC, 365
- RESTORE, 74
- RETREAT, 243
- RETREAT1, 243
- RETREAT2, 244
- REVEAL.TURTLE, 150
- REVERSE, 238
- RHYTHM, 239
- RING, 129
- ROAD, 118
- ROCKET, 132
- ROOM1, 199
- ROOM2, 200
- ROOM3, 200
- ROOM4, 200
- ROW, 72
- RUN.RACE, 210
  
- S, 129
- SATURN, 129
- SAVE.CURSOR, 74
- SAVEPICT, 290, 291
- SAVEPICT1, 290, 291
- SAVESH, 204
- SAVE.SHAPES, 170
- SAY, 37, 57
- SCALE, 375
- SCORE, 169
- SCRAMBLE, 72
- SCRAMBLE1, 72
- SEARCH, 361
- SECOND, 11
- SECOND.CLEANUP, 120
- SEGMENT, 57
- SEGMENTS, 57
  
- SELECT, 6
- SEN, 5
- SENDER'S.NAME, 66
- SEND.MAIL, 66
- SENGEN, 5
- SEQUENCER, 238
- SET, 320
- SET.ARGS, 321
- SETBOXES, 139
- SET.DEMONS, 150
- SET.DRIVE, 118
- SET.EGG, 103
- SETJOY, 140
- SET.MARKS, 102
- SET.MONSTER, 204
- SET.PIECES, 149
- SETPLACE, 129
- SETRING, 129
- SET.SCENE, 118
- SETSHAPES, 132
- SET.SHIP, 132
- SETUP, 36, 85, 117, 138, 149, 167, 369, 370
- SET.UP, 132
- SETUP.BLASTER, 158
- SETUP.BYE, 138
- SETUP.DEMONS, 158, 168, 198
- SETUP.ENEMIES, 158
- SETUP.GRAPHICS, 361
- SETUP.JJ, 121
- SETUP.LAND, 169
- SETUP.LAND.DEMONS, 169
- SETUP.PLAYER, 158
- SETUP.RACE, 210
- SETUP.RACERS, 210
- SETUP.SHAPES, 170
- SETUP.TURTLE, 256
- SETUP.VARS, 149
- SFUN?, 320
- SFUN.FUNC, 320
- SHOW.SCORE, 159
- SHOW.WINNER, 210
- SHOW.WORD, 73
- SHRINK, 367
- SIDE, 199
- SIGH, 86
- SING, 104
- SLOPE, 360
- SMILE, 38
- SOLID.SIDE, 199
- SOLVE, 361
- SOLVE.X, 346
- SOLVE.Y, 346
- SONG, 104, 238, 256
- SONG1, 238
- SOUND.DOWN, 130
- SOUND.OFF, 131
- SOUND.UP, 130
- SPACE.SHIP, 246
- SPEEDUP, 375
- SPELL, 50
- SPELL1, 50

- SPELL.GROUP, 49
- SPOTION, 203
- STAIRS, 201
- STAR, 129, 229, 364
- STAR3, 365
- STARS, 129
- START, 167, 198
- START.TURTLE, 361
- STAY.IN.BOUNDS, 150
- STEER, 159
- STEPVALUE, 247
- SUB1, 267
- SUB2, 45
- SUBMOUNTAIN, 130
- SUBTRACT, 45
- SWORD, 202
- SYMEVAL, 320
  
- TAKE, 189
- TAKEOFF, 103
- TAKE.OFF, 130
- TAKE.OFF.ACTION, 131
- TEEN, 50
- TEMPO, 257
- TENS, 49
- TESTWORD, 38
- THEEND, 103
- THREESTARS, 364
- TO1, 321
- TOUCHTOOT, 377
- TOWARDS, 213
- TOWARDS1, 213
  
- TOWARDS2, 213
- TOWARDS3, 213
- TRAMP.BED, 85
- TRAMP.LEGS, 85
- TRANSFORM, 120
- TRANSPPOSE.DOWN, 238
- TRANSPPOSE.UP, 238
- TRAP, 201
- TREES, 118
- TRIAD, 50
- TRILL, 247
- TRY.AGAIN, 242
- TRYDOOR, 189
- TUNE, 238
- TURTLESONG, 256
- TURTLETIMER, 379
  
- UFUN.ARGLIST, 320
- UFUN.BODY, 320
- UNBIND, 321
- UBIND.ARG, 321
- UBIND.ARGS, 321
- UNDOT, 320
- UNGALL, 38
- UNLOCK, 189
- UNQUOTE, 319
- UP, 245
- UPDATE, 150
- UP.THE.HILL, 121
- USE.SHAPES.WALK, 86
  
- VERB, 6
  
- VERBPHRASE, 6
- VERBS, 6
- VICTORY, 150
- VIEWPOLYGON, 220
- VIEWPOLYGON1, 220
- VOWELP, 78
- VSYM, 320
  
- W, 204
- WAFFLE, 363
- WALK, 86, 130, 169
- WALK.SOUND, 130
- WAND, 202
- WHIRL, 365
- WIN, 38, 121, 168
- WING, 103
- WIN.MESSAGES, 74
- WITHIN, 189
- WONP, 210
- WORDSCRAM, 71
  
- XDIFF, 291
- XLIST, 347
- XSQUARED, 346
- XTIMESY, 346
- XVALUE, 347
  
- YDIFF, 291
- YES, 151
- YES.BRANCH, 20
- YINTERCEPT, 346
- YLIST, 347